



Confluence by Z in Agda for PLFA

Vincent van Oostrom

University of Sussex

vvo@sussex.ac.uk

Pen-and-paper confluence of $\lambda\beta$ (cf. Barendregt 84)

Definition (λ -term; Church 32)

A λ -term either is a variable x or an application MN or a λ -abstraction $\lambda x.M$

Pen-and-paper confluence of $\lambda\beta$

Definition (λ -term)

A λ -term either is a variable x or an application MN or a λ -abstraction $\lambda x.M$

λ -terms **up to** α -congruence induced by $\lambda x.M = \lambda y.M[x:=y]$, for y **not in** M

Pen-and-paper confluence of $\lambda\beta$

Definition (λ -term)

A λ -term either is a variable x or an application MN or a λ -abstraction $\lambda x.M$

λ -terms up to α -congruence induced by $\lambda x.M = \lambda y.M[x:=y]$, for y **not in** M

Definition (β -reduction)

\rightarrow_β on λ -terms is **compatible closure** of β -scheme $(\lambda x.M) N = M[x:=N]$

Pen-and-paper confluence of $\lambda\beta$

Definition (λ -term)

A λ -term either is a variable x or an application MN or a λ -abstraction $\lambda x.M$

λ -terms up to α -congruence induced by $\lambda x.M = \lambda y.M[x:=y]$, for y **not in** M

Definition (β -reduction)

\rightarrow_β on λ -terms is compatible closure of β -scheme $(\lambda x.M)N = M[x:=N]$

$M[x:=N]$ the **capture-avoiding** substitution of N for x in M

Pen-and-paper confluence of $\lambda\beta$

Definition (λ -term)

A λ -term either is a variable x or an application MN or a λ -abstraction $\lambda x.M$

λ -terms up to α -congruence induced by $\lambda x.M = \lambda y.M[x:=y]$, for y **not in** M

Definition (β -reduction)

\rightarrow_β on λ -terms is compatible closure of β -scheme $(\lambda x.M) N = M[x:=N]$

$M[x:=N]$ the capture-avoiding substitution of N for x in M

Theorem (Church–Rosser 36)

\rightarrow_β has the Church–Rosser property

Pen-and-paper confluence of $\lambda\beta$

Definition (λ -term)

A λ -term either is a variable x or an application MN or a λ -abstraction $\lambda x.M$

λ -terms up to α -congruence induced by $\lambda x.M = \lambda y.M[x:=y]$, for y **not in** M

Definition (β -reduction)

\rightarrow_β on λ -terms is compatible closure of β -scheme $(\lambda x.M) N = M[x:=N]$

$M[x:=N]$ the capture-avoiding substitution of N for x in M

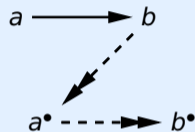
Theorem (Church–Rosser)

\rightarrow_β has the Church–Rosser property

$\iff \rightarrow_\beta$ is confluent $\iff \twoheadrightarrow_\beta$ has the diamond property

Z

Definition (Z-property of \rightarrow for bullet-function \bullet on objects)



Z

Definition (Z-property of \rightarrow for \bullet ; Loader 98, Dehornoy & \heartsuit 08)

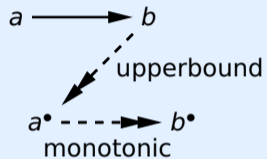
for every step $a \rightarrow b$

(upperbound) $b \twoheadrightarrow a^\bullet$

(monotonic) $a^\bullet \twoheadrightarrow b^\bullet$

Z

Definition (Z-property of \rightarrow for \bullet)

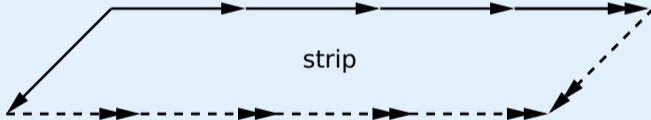


Z

Definition (Z-property of \rightarrow for \bullet)

for every step $a \rightarrow b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma ($Z \implies \text{strip} \implies \text{confluence}$ (Barendregt 84))

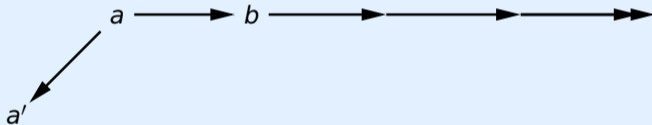


Z

Definition (Z-property of \rightarrow for \bullet)

for every step $a \rightarrow b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (**Z** \implies **strip** \implies **confluence**)

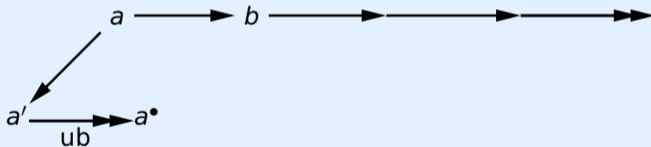


Z

Definition (Z-property of \rightarrow for \bullet)

for every step $a \rightarrow b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (**Z** \implies **strip** \implies **confluence**)

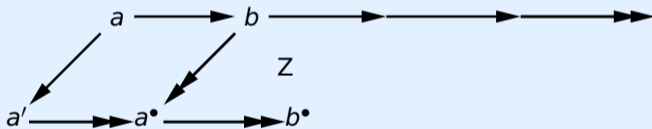


Z

Definition (Z-property of \rightarrow for \bullet)

for every step $a \rightarrow b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (**Z** \implies **strip** \implies **confluence**)

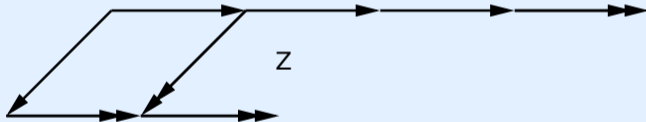


Z

Definition (Z-property of \rightarrow for \bullet)

for every step $a \rightarrow b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (**Z** \implies strip \implies confluence)

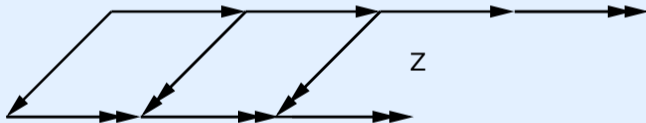


Z

Definition (Z-property of \rightarrow for \bullet)

for every step $a \rightarrow b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (**Z** \implies **strip** \implies **confluence**)

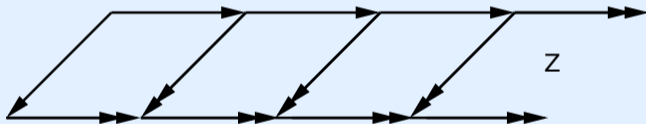


Z

Definition (Z-property of \rightarrow for \bullet)

for every step $a \rightarrow b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (**Z** \implies **strip** \implies **confluence**)

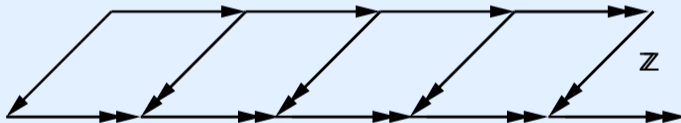


Z

Definition (Z-property of \rightarrow for \bullet)

for every step $a \rightarrow b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (**Z** \implies **strip** \implies **confluence**)

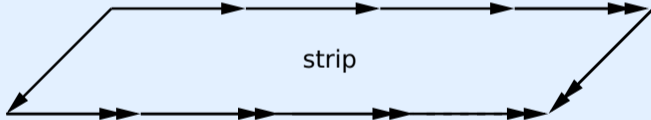


Z

Definition (Z-property of \rightarrow for \bullet)

for every step $a \rightarrow b$, both $b \twoheadrightarrow a^\bullet$ (ub) and $a^\bullet \twoheadrightarrow b^\bullet$ (mon)

Lemma (Z \implies strip \implies confluence)



Z for $\lambda\beta$

Theorem (Loader 98)

for every step $a \rightarrow_{\beta} b$, both $b \rightarrow_{\beta} a^{\bullet}$ (ub) and $a^{\bullet} \rightarrow_{\beta} b^{\bullet}$ (mon), where

$$x^{\bullet} := x$$

$$(\lambda x.M)^{\bullet} := \lambda x.M^{\bullet}$$

$$((\lambda x.M) N)^{\bullet} := M^{\bullet}[x:=N^{\bullet}]$$

$$(MN)^{\bullet} := M^{\bullet} N^{\bullet} \quad \text{otherwise (if } M = x \text{ or } M = PQ)$$

Z for $\lambda\beta$

Theorem (Loader 98)

for every step $a \rightarrow_{\beta} b$, both $b \rightarrow_{\beta} a^{\bullet}$ (ub) and $a^{\bullet} \rightarrow_{\beta} b^{\bullet}$ (mon), where

$$x^{\bullet} := x$$

$$(\lambda x.M)^{\bullet} := \lambda x.M^{\bullet}$$

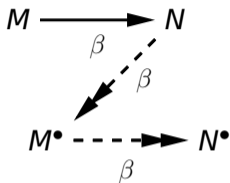
$$((\lambda x.M) N)^{\bullet} := M^{\bullet}[x:=N^{\bullet}]$$

$$(MN)^{\bullet} := M^{\bullet} N^{\bullet} \quad \textit{otherwise}$$

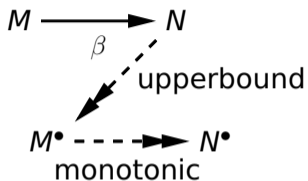
Remark

full development map • contracting **all** β -redex-patterns in λ -term
(Church–Rosser 30s; Gross–Knuth, **preprint** 70s; Takahashi, Loader 90s)

Z for $\lambda\beta$ proof



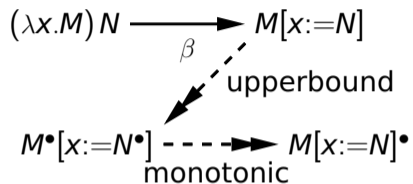
Z for $\lambda\beta$ proof



Proof.

(ub) and (mon) **by induction** on $M \rightarrow_{\beta} N$

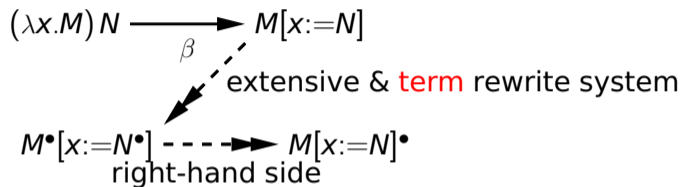
Z for $\lambda\beta$ proof



Proof.

(ub) and (mon) by induction on $M \rightarrow_\beta N$ with **base case** β

Z for $\lambda\beta$ proof



Proof.

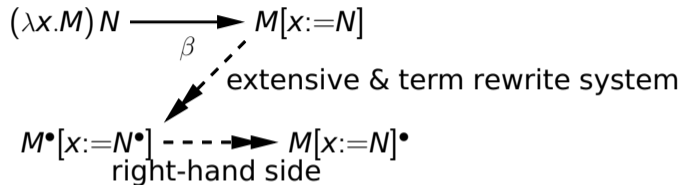
(ub) and (mon) by induction on $M \rightarrow_\beta N$ with base case β , using:

(extensive) $M \rightarrow_\beta M^\bullet$

(ctx, sub) if $M \rightarrow_\beta N$ and $P \rightarrow_\beta Q$, then $M[x:=P] \rightarrow_\beta N[y:=Q]$

(right-hand side) $M^\bullet[x:=N^\bullet] \rightarrow_\beta M[x:=N]^\bullet$

Z for $\lambda\beta$ proof



Proof.

(ub) and (mon) by induction on $M \rightarrow_\beta N$ with base case β , using:

(extensive) $M \rightarrow_\beta M^\bullet$

($\overline{\text{ctx}}, \overline{\text{sub}}$) if $M \rightarrow_\beta N$ and $P \rightarrow_\beta Q$, then $M[x:=P] \rightarrow_\beta N[y:=Q]$

(right-hand side) $M^\bullet[x:=N^\bullet] \rightarrow_\beta M[x:=N]^\bullet$

(ext), (rhs), ($\overline{\text{ctx}}$) by induction on M ; ($\overline{\text{sub}}$) by induction on $M \rightarrow_\beta N$ □

Substitution lemma

Lemma (β -critical peak)

$((\lambda x.M) N)[y:=Q] \beta\leftarrow (\lambda y.(\lambda x.M) N) Q \rightarrow_{\beta} (\lambda y.M[x:=N]) Q$ is single-step joinable

Substitution lemma

Lemma (β -critical peak)

$((\lambda x.M) N)[y:=Q] \beta\leftarrow (\lambda y.(\lambda x.M) N) Q \rightarrow_{\beta} (\lambda y.M[x:=N]) Q$ is single-step joinable

Proof.

$((\lambda x.M) N)[y:=Q] = (\lambda x.M[y:=Q]) N[y:=Q] \rightarrow_{\beta}$
 $M[y:=Q][x:=N[y:=Q]] =_{SL} M[x:=N][y:=Q] \beta\leftarrow (\lambda y.M[x:=N]) Q$ □

Substitution lemma

Lemma (β -critical peak)

$((\lambda x.M) N)[y:=Q] \beta\leftarrow (\lambda y.(\lambda x.M) N) Q \rightarrow_{\beta} (\lambda y.M[x:=N]) Q$ is single-step joinable

Proof.

$((\lambda x.M) N)[y:=Q] = (\lambda x.M[y:=Q]) N[y:=Q] \rightarrow_{\beta}$
 $M[y:=Q][x:=N[y:=Q]] =_{SL} M[x:=N][y:=Q] \beta\leftarrow (\lambda y.M[x:=N]) Q$ □

Remark

closure of \rightarrow_{β} under substitution ($\overline{\text{sub}}$) \iff β -critical peak lemma \iff SL

Substitution lemma

Lemma (β -critical peak)

$((\lambda x.M) N)[y:=Q] \beta\leftarrow (\lambda y.(\lambda x.M) N) Q \rightarrow_{\beta} (\lambda y.M[x:=N]) Q$ is single-step joinable

Proof.

$((\lambda x.M) N)[y:=Q] = (\lambda x.M[y:=Q]) N[y:=Q] \rightarrow_{\beta}$
 $M[y:=Q][x:=N[y:=Q]] =_{SL} M[x:=N][y:=Q] \beta\leftarrow (\lambda y.M[x:=N]) Q$ □

Remark

closure of \rightarrow_{β} under substitution ($\overline{\text{sub}}$) \iff β -critical peak lemma \iff SL
proof of (rhs) uses substitution lemma

Substitution lemma

Lemma (β -critical peak)

$((\lambda x.M) N)[y:=Q] \beta\leftarrow (\lambda y.(\lambda x.M) N) Q \rightarrow_{\beta} (\lambda y.M[x:=N]) Q$ is single-step joinable

Proof.

$((\lambda x.M) N)[y:=Q] = (\lambda x.M[y:=Q]) N[y:=Q] \rightarrow_{\beta}$
 $M[y:=Q][x:=N[y:=Q]] =_{SL} M[x:=N][y:=Q] \beta\leftarrow (\lambda y.M[x:=N]) Q$ □

Remark

closure of \rightarrow_{β} under substitution ($\overline{\text{sub}}$) \iff β -critical peak lemma \iff SL
proof of (rhs) uses substitution lemma
 β -redexes **do** have overlap (redex-patterns do not)

Substitution lemma

Lemma (β -critical peak)

$((\lambda x.M) N)[y:=Q] \beta\leftarrow (\lambda y.(\lambda x.M) N) Q \rightarrow_{\beta} (\lambda y.M[x:=N]) Q$ is single-step joinable

Proof.

$((\lambda x.M) N)[y:=Q] = (\lambda x.M[y:=Q]) N[y:=Q] \rightarrow_{\beta}$
 $M[y:=Q][x:=N[y:=Q]] =_{SL} M[x:=N][y:=Q] \beta\leftarrow (\lambda y.M[x:=N]) Q$ □

Remark

closure of \rightarrow_{β} under substitution ($\overline{\text{sub}}$) $\iff \beta$ -critical peak lemma \iff SL
proof of (rhs) **uses** substitution lemma

β -redexes do have overlap; SL **needed** to have **term** rewrite system

Formalisation of confluence by Z for $\lambda\beta$

Motivation

confluence of $\lambda\beta$ -calculus PL-**litmus** test for proof assistants
(inductive λ -terms and β -steps, binding, substitution, modulo α)

Formalisation of confluence by Z for $\lambda\beta$

Motivation

confluence of $\lambda\beta$ -calculus PL-litmus test for proof assistants

claim: Z gives **shortest** proof of confluence of $\lambda\beta$

Formalisation of confluence by Z for $\lambda\beta$ in Agda

Motivation

confluence of $\lambda\beta$ -calculus PL-litmus test for proof assistants

claim: Z gives shortest proof of confluence of $\lambda\beta$

here: test claim in Agda; had wanted to learn some Agda for some time
(done in 2021; learned at IWC 2023 of Andrea Laretto's Ba thesis)

Formalisation of confluence by Z in Agda based on PLFA

Motivation

confluence of $\lambda\beta$ -calculus PL-litmus test for proof assistants

claim: Z gives shortest proof of confluence of $\lambda\beta$

here: test claim in Agda

design decision: adapt extant PLFA proof (by **triangle** property; Takahashi 95)
(Programming Language Foundations in Agda by Wadler, Kokke, Siek 20
adaptation allowed reuse of inductive λ -terms, β -steps and SL
reuse good software engineering and useful since absolute Agda beginner)

Formalisation of confluence by Z in Agda based on PLFA

Motivation

confluence of $\lambda\beta$ -calculus PL-litmus test for proof assistants

claim: Z gives shortest proof of confluence of $\lambda\beta$

here: test claim in Agda

design decision: adapt extant PLFA proof (by triangle property

also means have to stick with design decisions of PLFA

λ -terms in PLFA

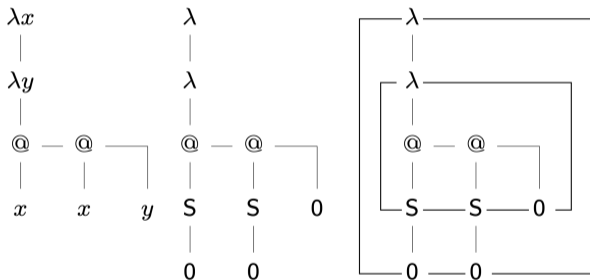
Definition (Nameless λ -term; de Bruijn 72)

PLFA design decision: scoped **nameless** λ -terms
(avoids α -renaming at the expense of **re-indexing**)

λ -terms in PLFA

Definition (Nameless λ -term)

PLFA design decision: scoped nameless λ -terms



2: **named** $\lambda x. \lambda y. x (x y)$, **nameless** $\lambda \lambda S 0 ((S 0) 0)$, **scoped** $0 \vdash \lambda \lambda S 0 ((S 0) 0)$

λ -terms in PLFA

Definition (Nameless λ -term)

PLFA design decision: **scoped** nameless λ -terms

Definition (Scoped λ -term; \forall & van der Looij & Zwitserlood 04??)

$i \vdash t$ is nameless λ -term t in **scope** i

(think of i as binding-**stack** with t closed within it; i is upperbound on indices in t)

λ -terms in PLFA

Definition (Nameless λ -term)

PLFA design decision: scoped nameless λ -terms

Definition (Scoped λ -term)

$i \vdash t$ is nameless λ -term t in scope i (bottom-up) **inductively derivable** by:

$$\frac{Si \vdash 0}{0} \quad 0 \quad \frac{Si \vdash St}{i \vdash t} S \quad \frac{i \vdash \lambda t}{Si \vdash t} \lambda \quad \frac{i \vdash t_1 t_2}{i \vdash t_1 \quad i \vdash t_2} @$$

λ -terms in PLFA

Definition (Nameless λ -term)


PLFA design decision: scoped nameless λ -terms

Definition (Scoped λ -term)

$i \vdash t$ is nameless λ -term t in scope i

$$\frac{Si \vdash 0}{0} \quad \frac{Si \vdash St}{i \vdash t} S \quad \frac{i \vdash \lambda t}{Si \vdash t} \lambda \quad \frac{i \vdash t_1 t_2}{i \vdash t_1 \quad i \vdash t_2} @$$

Remark

these are **generalised** nameless λ -terms (Bird & Paterson 99; Hendriks &  03)
PLFA only allows S on 0 and on other Ss; nameless λ -terms; no ho-signature

\rightarrow_{β} -steps in PLFA

Definition (Nameless β -reduction)

PLFA design decision: **single** substitution $t[s]$ by **parallel** substitution $0 \mapsto s, Si \mapsto i$
(substitute s for the **free** 0 s in t ; **decrement** other indices)

\rightarrow_{β} -steps in PLFA

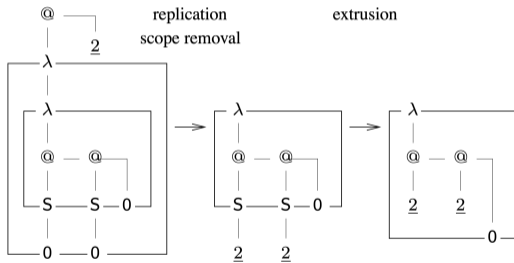
Definition (Nameless β -reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
 \rightarrow_{β} on nameless λ -terms is **compatible closure** of β -scheme $i \vdash (\lambda t) s = i \vdash t[s]$

\rightarrow_{β} -steps in PLFA

Definition (Nameless β -reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
 \rightarrow_{β} on nameless λ -terms is compatible closure of β -scheme $i \vdash (\lambda t) s = i \vdash t[s]$



$$0 \vdash \underline{2} \underline{2} \rightarrow_{\beta} 0 \vdash (\lambda S 0 ((S 0) 0))[\underline{2}] = 0 \vdash \lambda S \underline{2} ((S \underline{2}) 0) = 0 \vdash \lambda \underline{2} (\underline{2} 0)$$

\rightarrow_{β} -steps in PLFA

Definition (Nameless β -reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
 \rightarrow_{β} on nameless λ -terms is compatible closure of β -scheme $i \vdash (\lambda t) s = i \vdash t[s]$

Lemma ($\lambda\beta$ is a ho-term rewriting system; in PLFA)

$(\overline{ctx}, \overline{sub})$ closure of reduction under contexts, substitutions

(substitution lemma) for single substitution *via* parallel substitution

\rightarrow_{β} -steps in PLFA

Definition (Nameless β -reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
 \rightarrow_{β} on nameless λ -terms is compatible closure of β -scheme $i \vdash (\lambda t) s = i \vdash t[s]$

Lemma ($\lambda\beta$ is a ho-term rewriting system; in PLFA)

$(\overline{ctx}, \overline{sub})$ closure of reduction under contexts, substitutions

(substitution lemma) for single substitution via parallel substitution

Remark

(\overline{ctx}) is called **congruence** in PLFA (wrong; **compatible**)

\rightarrow_{β} -steps in PLFA

Definition (Nameless β -reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
 \rightarrow_{β} on nameless λ -terms is compatible closure of β -scheme $i \vdash (\lambda t) s = i \vdash t[s]$

Lemma (λ_{β} is a ho-term rewriting system; in PLFA)

$(\overline{ctx}, \overline{sub})$ closure of reduction under contexts, substitutions

(substitution lemma) for single substitution via parallel substitution

Remark

(\overline{ctx}) is called congruence in PLFA; (\overline{sub}) **missing** from PLFA (50 loc)

\rightarrow_{β} -steps in PLFA

Definition (Nameless β -reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
 \rightarrow_{β} on nameless λ -terms is compatible closure of β -scheme $i \vdash (\lambda t) s = i \vdash t[s]$

Lemma ($\lambda\beta$ is a ho-term rewriting system; in PLFA)

$(\overline{ctx}, \overline{sub})$ closure of reduction under contexts, substitutions

(substitution lemma) for single substitution via parallel substitution

Remark

(\overline{ctx}) is called congruence in PLFA; (\overline{sub}) missing from PLFA; (substitution lemma) is called **commutation** in PLFA (wrong; **self-distributivity** / **associativity**)

\rightarrow_{β} -steps in PLFA

Definition (Nameless β -reduction)

PLFA design decision: single substitution $t[s]$ by parallel substitution $0 \mapsto s, Si \mapsto i$
 \rightarrow_{β} on nameless λ -terms is compatible closure of β -scheme $i \vdash (\lambda t) s = i \vdash t[s]$

Lemma ($\lambda\beta$ is a ho-term rewriting system; in PLFA)

$(\overline{ctx}, \overline{sub})$ closure of reduction under contexts, substitutions

(substitution lemma) for single substitution via parallel substitution

Remark

(\overline{ctx}) is called congruence in PLFA; (\overline{sub}) missing from PLFA; (substitution lemma) is called commutation in PLFA; terms / steps ad hoc (no signature)

Basic rewriting and full development map • for PLFA

app-cong : $\forall\{\Gamma\} \{K L M N : \Gamma \vdash \star\} \rightarrow K \longrightarrow L \rightarrow M \longrightarrow N \rightarrow K \cdot M \longrightarrow L \cdot N$

rew-rew : $\forall\{\Gamma\} \{M N : \Gamma, \star \vdash \star\} \{K L : \Gamma \vdash \star\}$

$\rightarrow M \longrightarrow N$

$\rightarrow K \longrightarrow L$

$\rightarrow M [K] \longrightarrow N [L]$

Remark

$\{\dots\}$ indicates implicit argument; Γ is scope; \star is singleton type of λ -terms

app-cong function taking reductions $K \rightarrow_{\beta} L$ and $M \rightarrow_{\beta} N$ yielding $K M \rightarrow_{\beta} L N$

rew-rew same but yielding closure under contexts, substitutions

Basic rewriting and full development map • for PLFA

$\text{app-cong} : \forall \{ \Gamma \} \{ K L M N : \Gamma \vdash \star \} \rightarrow K \longrightarrow L \rightarrow M \longrightarrow N \rightarrow K \cdot M \longrightarrow L \cdot N$

$\text{rew-rew} : \forall \{ \Gamma \} \{ M N : \Gamma , \star \vdash \star \} \{ K L : \Gamma \vdash \star \}$

$\rightarrow M \longrightarrow N$

$\rightarrow K \longrightarrow L$

$\rightarrow M [K] \longrightarrow N [L]$

$_ \bullet : \forall \{ \Gamma A \} \rightarrow \Gamma \vdash A \rightarrow \Gamma \vdash A$

$(' x) \bullet = ' x$

$(\lambda M) \bullet = \lambda (M \bullet)$

$((\lambda M) \cdot N) \bullet = M \bullet [N \bullet]$

$(M \cdot N) \bullet = (M \bullet) \cdot (N \bullet)$

prime indicates index (as λ -term)

(Extensive) $M \rightarrow M^\bullet$ for PLFA

$\text{extensive} : \forall \{ \Gamma A \} \rightarrow (M : \Gamma \vdash A) \rightarrow M \longrightarrow M \bullet$

$\text{extensive} (' _) = _ \blacksquare$

$\text{extensive} (\lambda M) = \text{abs-cong} (\text{extensive } M)$

$\text{extensive} ((\lambda M) \cdot N) = _ \longrightarrow \langle \beta \rangle \text{rew-rew} (\text{extensive } M) (\text{extensive } N)$

$\text{extensive} (' _ \cdot N) = \text{appR-cong} (\text{extensive } N)$

$\text{extensive} (L \cdot M \cdot N) = \text{app-cong} (\text{extensive } (L \cdot M)) (\text{extensive } N)$

(Extensive) $M \rightarrow M^\bullet$ for PLFA

extensive : $\forall \{ \Gamma A \} \rightarrow (M : \Gamma \vdash A) \rightarrow M \longrightarrow M \bullet$

extensive (' $_$) = $_ \blacksquare$

extensive (λM) = **abs-cong** (**extensive** M)

extensive ($(\lambda M) \cdot N$) = $_ \longrightarrow \langle \beta \rangle$ **rew-rew** (**extensive** M) (**extensive** N)

extensive (' $_ \cdot N$) = **appR-cong** (**extensive** N)

extensive ($L \cdot M \cdot N$) = **app-cong** (**extensive** ($L \cdot M$)) (**extensive** N)

Remark

recursion on scoped nameless $M : \Gamma \vdash A$ (\blacksquare is empty reduction)

otherwise only compatibility (wrongly named congruence in PLFA)

(Upperbound) $N \rightarrow M^\bullet$ if $M \rightarrow_\beta N$ for PLFA

upperbound : $\forall \{\Gamma\} \rightarrow \{M N : \Gamma \vdash \star\}$

$\rightarrow M \longrightarrow N$

$\rightarrow N \longrightarrow M^\bullet$

upperbound $\{_ \}$ $\{\lambda _ \}$ $(\zeta \phi) = \text{abs-cong (upperbound } \phi)$

upperbound $\{_ \}$ $\{(_ _) \cdot _ \}$ $\{_ \}$ $(\xi_2 \phi) = \text{appR-cong (upperbound } \phi)$

upperbound $\{_ \}$ $\{(\lambda _) \cdot M\}$ $\{((\lambda _) \cdot M)\}$ $(\xi_1 (\zeta \phi)) = _ \longrightarrow \langle \beta \rangle \text{rew-rew (upperbound } \phi) (\text{extensive } M)$

upperbound $\{_ \}$ $\{(\lambda L) \cdot _ \}$ $\{((\lambda L) \cdot _ \)}$ $(\xi_2 \phi) = _ \longrightarrow \langle \beta \rangle \text{rew-rew (extensive } L) (\text{upperbound } \phi)$

upperbound $\{_ \}$ $\{(\lambda L) \cdot M\}$ $\{.(\text{subst (subst-zero } M) L)\}$ $\beta = \text{rew-rew (extensive } L) (\text{extensive } M)$

upperbound $\{_ \}$ $\{_ \cdot _ \cdot M\}$ $\{.(_ \cdot M)\}$ $(\xi_1 \phi) = \text{app-cong (upperbound } \phi) (\text{extensive } M)$

upperbound $\{_ \}$ $\{K \cdot L \cdot _ \}$ $\{.(_ \cdot _ \cdot _ \)}$ $(\xi_2 \phi) = \text{app-cong (extensive } (K \cdot L)) (\text{upperbound } \phi)$

(Upperbound) $N \rightarrow M^\bullet$ if $M \rightarrow_\beta N$ for PLFA

upperbound : $\forall \{\Gamma\} \rightarrow \{M N : \Gamma \vdash \star\}$

$\rightarrow M \longrightarrow N$

$\rightarrow N \longrightarrow M^\bullet$

upperbound $\{_ \}$ $\{\lambda _ \}$ $(\zeta \phi) = \text{abs-cong (upperbound } \phi)$

upperbound $\{_ \}$ $\{(_ _)\cdot _ \}$ $\{_ \}$ $(\xi_2 \phi) = \text{appR-cong (upperbound } \phi)$

upperbound $\{_ \}$ $\{(\lambda _)\cdot M\}$ $\{((\lambda _)\cdot M)\}$ $(\xi_1 (\zeta \phi)) = _ \longrightarrow \langle \beta \rangle \text{rew-rew (upperbound } \phi) (\text{extensive } M)$

upperbound $\{_ \}$ $\{(\lambda L)\cdot _ \}$ $\{((\lambda L)\cdot _ \)}$ $(\xi_2 \phi) = _ \longrightarrow \langle \beta \rangle \text{rew-rew (extensive } L) (\text{upperbound } \phi)$

upperbound $\{_ \}$ $\{(\lambda L)\cdot M\}$ $\{.(\text{subst (subst-zero } M) L)\}$ $\beta = \text{rew-rew (extensive } L) (\text{extensive } M)$

upperbound $\{_ \}$ $\{_ \cdot _ \cdot M\}$ $\{.(_ \cdot M)\}$ $(\xi_1 \phi) = \text{app-cong (upperbound } \phi) (\text{extensive } M)$

upperbound $\{_ \}$ $\{K \cdot L \cdot _ \}$ $\{.(_ \cdot _ \cdot _ \)}$ $(\xi_2 \phi) = \text{app-cong (extensive } (K \cdot L)) (\text{upperbound } \phi)$

Remark

recursion on $M \rightarrow_\beta N$ (ζ, ξ_1, ξ_2 traditional names of compatibility clauses)
otherwise only (extensive) and compatibility

(Right-hand side) $(M^\bullet)^{\sigma^\bullet} \rightarrow (M^\sigma)^\bullet$ for PLFA

$\text{rhss} : \forall\{\Gamma \Delta\} (M : \Gamma \vdash \star) \{\sigma \tau : \text{Subst } \Gamma \Delta\} \rightarrow ((x : \Gamma \ni \star) \rightarrow \tau x \equiv \sigma x \bullet)$

 $\rightarrow \text{subst } \tau (M \bullet) \longrightarrow (\text{subst } \sigma M) \bullet$

$\text{rhss } ('x) \text{ eq rewrite } (eq x) = _ \blacksquare$

$\text{rhss } (\lambda M) \text{ eq} = \text{abs-cong } (\text{rhss } M \text{ (exts-bullet eq)})$

$\text{rhss } (('x) \cdot M) \{\sigma\} \text{ eq rewrite } (eq x) = \longrightarrow\text{-trans}$

$(\text{appR-cong } (\text{rhss } M \text{ eq})) (\text{app-bullet } (\sigma x) (\text{subst } \sigma M)) \text{ where}$

$\{- \text{auxiliary rhs/monotonicity lemma for application -}\}$

$\text{app-bullet} : \forall\{\Gamma\} (L M : \Gamma \vdash \star) \rightarrow L \bullet \cdot M \bullet \longrightarrow (L \cdot M) \bullet$

$\text{app-bullet } ('_) _ = _ \blacksquare$

$\text{app-bullet } (\lambda _) _ = (_ \longrightarrow \langle \beta \rangle _) \blacksquare$

$\text{app-bullet } (_ \cdot _) _ = _ \blacksquare$

$\text{rhss } ((\lambda L) \cdot M) \{\tau = \tau\} \text{ eq rewrite } (\text{sym } (\text{subst-commute } \{N = L \bullet\} \{M \bullet\} \{\tau\})) =$

$\text{rew-rew } (\text{rhss } L \text{ (exts-bullet eq)}) (\text{rhss } M \text{ eq})$

$\text{rhss } (K \cdot L \cdot M) \text{ eq} = \text{app-cong } (\text{rhss } (K \cdot L) \text{ eq}) (\text{rhss } M \text{ eq})$

(Right-hand side) $(M^\bullet)^\sigma \rightarrow (M^\sigma)^\bullet$ for PLFA

$\text{rhss} : \forall\{\Gamma \Delta\} (M : \Gamma \vdash \star) \{\sigma \tau : \text{Subst } \Gamma \Delta\} \rightarrow ((x : \Gamma \ni \star) \rightarrow \tau x \equiv \sigma x)^\bullet$

 $\rightarrow \text{subst } \tau (M \bullet) \longrightarrow (\text{subst } \sigma M)^\bullet$

$\text{rhss } ('x) \text{ eq rewrite } (eq x) = _ \blacksquare$

$\text{rhss } (\lambda M) \text{ eq} = \text{abs-cong } (\text{rhss } M \text{ (exts-bullet eq)})$

$\text{rhss } (('x) \cdot M) \{\sigma\} \text{ eq rewrite } (eq x) = \longrightarrow\text{-trans}$

$(\text{appR-cong } (\text{rhss } M \text{ eq})) (\text{app-bullet } (\sigma x) (\text{subst } \sigma M)) \text{ where}$

$\{- \text{auxiliary rhs/monotonicity lemma for application -}\}$

$\text{app-bullet} : \forall\{\Gamma\} (L M : \Gamma \vdash \star) \rightarrow L \bullet \cdot M \bullet \longrightarrow (L \cdot M)^\bullet$

$\text{app-bullet } ('_) _ = _ \blacksquare$

$\text{app-bullet } (\lambda _) _ = (_ \longrightarrow \langle \beta \rangle _) \blacksquare$

$\text{app-bullet } (_ \cdot _) _ = _ \blacksquare$

$\text{rhss } ((\lambda L) \cdot M) \{\tau = \tau\} \text{ eq rewrite } (\text{sym } (\text{subst-commute } \{N = L \bullet\} \{M \bullet\} \{\tau\})) =$

$\text{rew-rew } (\text{rhss } L \text{ (exts-bullet eq)}) (\text{rhss } M \text{ eq})$

$\text{rhss } (K \cdot L \cdot M) \text{ eq} = \text{app-cong } (\text{rhss } (K \cdot L) \text{ eq}) (\text{rhss } M \text{ eq})$

Remark

recursion on scoped nameless $M : \Gamma \vdash \star$ (σ, τ are parallel substitutions)

(Monotonic) $M^\bullet \twoheadrightarrow_\beta N^\bullet$ if $M \twoheadrightarrow_\beta N$ for PLFA

monotonic : $\forall \{\Gamma\} \rightarrow \{M N : \Gamma \vdash \star\}$

$\rightarrow M \longrightarrow N$

$\rightarrow M^\bullet \longrightarrow N^\bullet$

monotonic $(\zeta \phi) = \text{abs-cong} (\text{monotonic } \phi)$

monotonic $\{_ \} \{(_ _)\cdot _ \} \{(_ _)\cdot _ \} (\xi_2 \phi) = \text{appR-cong} (\text{monotonic } \phi)$

monotonic $\{\Gamma\} \{(\lambda M) \cdot N\} \{.(subst (subst-zero N) M)\} \beta = \text{rhss } M \text{ bullet-zero where}$

$\{- \text{bullet commutes with lifting terms to substitutions} -\}$

bullet-zero : $(x : \Gamma, \star \ni \star) \rightarrow \text{subst-zero } (N \cdot) x \equiv \text{subst-zero } N x \cdot$

bullet-zero Z = refl

bullet-zero (S x) = refl

monotonic $\{_ \} \{(\lambda _)\cdot _ \} \{(\lambda _)\cdot _ \} (\xi_1 (\zeta \phi)) = \text{rew-rew} (\text{monotonic } \phi) (_ \blacksquare)$

monotonic $\{_ \} \{(\lambda M) \cdot _ \} \{.((\lambda M) \cdot _)\} (\xi_2 \phi) = \text{rew-rew} (M \cdot \blacksquare) (\text{monotonic } \phi)$

monotonic $\{_ \} \{_ \cdot _ \cdot _ \} \{(\lambda _)\cdot _ \} (\xi_1 \phi) = \text{--->-trans} (\text{appL-cong} (\text{monotonic } \phi)) (_ \text{--->}\langle \beta \rangle _ \blacksquare)$

monotonic $\{_ \} \{_ \cdot _ \cdot _ \} \{(_ _)\cdot _ \} (\xi_1 \phi) = \text{appL-cong} (\text{monotonic } \phi)$

monotonic $\{_ \} \{_ \cdot _ \cdot _ \} \{_ \cdot _ \cdot _ \} (\xi_1 \phi) = \text{appL-cong} (\text{monotonic } \phi)$

monotonic $\{_ \} \{_ \cdot _ \cdot _ \} \{_ \cdot _ \cdot _ \} (\xi_2 \phi) = \text{appR-cong} (\text{monotonic } \phi)$

Conclusions

- 4 key properties 65 loc

induction on scoped nameless λ -term		induction on derivations
(extensive)	\implies	(upperbound)
(right-hand side)	\implies	(monotonic)


- nameless = **un**named; scopes are stacks (Hendriks & \heartsuit 03; not lists)
- basic term rewrite theory of $\lambda\beta$ in PLFA is incomplete
(no signature; does not show it's a ho-term rewrite system, no $\overline{\text{sub}}$)
- section on confluence of $\lambda\beta$ in PLFA is suboptimal
(shorter proof via Z; the notes attributing are incorrect / improper)

Future work

Questions

- single proof **instantiating** to full development, full superdevelopment maps?

Remark

full superdevelopment map • contracting β -redex-patterns in inside-out sweep (Aczel 80s; van Raamsdonk 90s, Dehornoy &  00s)

Future work

Questions

- single proof instantiating to full development, **full superdevelopment** maps?

if $a \rightarrow_{\beta} b$ then $b \rightarrow_{\beta} a^{\bullet} \rightarrow_{\beta} b^{\bullet}$ (Z; Dehornoy & 08), where

$$x^{\bullet} := x$$

$$(\lambda x.M)^{\bullet} := \lambda x.M^{\bullet}$$

$$(MN)^{\bullet} := M'[x:=N^{\bullet}] \quad \text{if } M^{\bullet} = \lambda x.M'$$

$$:= M^{\bullet} N^{\bullet} \quad \text{otherwise}$$

Future work

Questions

- single proof instantiating to full development, full superdevelopment maps?
- avoid duplicates of **substitution** lemmata in PLFA? (for **reindexing**)


Future work

Questions

- single proof instantiating to full development, full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?

by working with **generalised** scoped λ -terms (instead of separate **indices**)

Remark

generalised scoped λ -terms due to Bird & Paterson 99, Hendriks &  02 & van der Looij & Zwitterlood 04

Future work

Questions

- single proof instantiating to full development, full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?
- avoid parallel substitution in PLFA? (only **single** substitution)

Future work

Questions

- single proof instantiating to full development, full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?
- avoid parallel substitution in PLFA?

by working with single substitution **at a given depth**

Remark

analogous to Huet's 94 Coq formalisation (based on 6 axioms); cf. proceedings

Future work

Questions

- single proof instantiating to full development, full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?
- avoid parallel substitution in PLFA?
- avoid maximal scope extrusion? (work with **minimal** scope extrusion)

Remark

analogous to Hendriks &  03; cf. paper in proceedings

Future work

Questions

- single proof instantiating to full development, full superdevelopment maps?
- avoid duplicates of substitution lemmata in PLFA?
- avoid parallel substitution in PLFA?
- avoid maximal scope extrusion?

Thanks to

Patrick Dehornoy, Christian Sternagel, Julian Nagele, Bertram Felgenhauer for Z
Andrea Laretto 24 for discussing confluence by Z in Agda (for PLFA; ongoing)